characterized in that the kernel constrains context switching to occur only at task level, rather than allowing context switches at lower sub-routine level.

2. The RTOS of claim 1 wherein the RTOS operates with a single call-return stack common to all of the defined tasks.

3. The RTOS of claim 2 wherein the single stack is implemented as a general-purpose stack.

4. The RTOS of claim 2 wherein the single stack is implemented as a hardware call...return stack.

5. The RTOS of claim 2 comprising a specific task control block assigned to each task, wherein a single task-resume address is saved.

6. The RTOS of claim 5 wherein additional task-specific information is saved.

7. The RTOS of claim 5 wherein a task-resume address is obtained in a context switch by placing a label at the point where the task is to resume, and obtaining the address of the label and storing that address as the task-resume address.

8. The RTOS of claim 7 wherein multiple labels are used within a single task to accomplish multiple context switches.

9. The RTOS of claim 1 further comprising a wait-on-event function characterized in that the function is called only at task-level, returns a value based on whether an event is available or not, and initiates a context switch or not based

on the returned value.

10. The RTOS of claim 1 further comprising a wait-on-event function enclosed within a (while) loop at task level, and characterized in that the task calls the wait-on-event function in the loop and examines its return code, exiting the loop if the event is available and initiates a context switch if not, and in the event of a context switch, the task recalls the wait-on-event function after resumption, being still in the loop, and repeats this procedure until exiting the loop.

11. A method for operating a minimal-memory controller comprising steps of:

(a) executing by the controller a real-time operating system (RTOS) based on kernel-controlled multitasking;

(b) calling defined tasks by the kernel, with individual ones of the tasks calling component subroutines; and

(c) constraining context-switching to occur solely at the task level rather than at any lower sub-routine level.

12. The method of claim 11 wherein the RTOS operates with a single call-return stack common to all of the defined tasks.

13. The method of claim 12 wherein the single stack is implemented as a general-purpose stack.

14. The method of claim 12 wherein the single stack is implemented as a hardware call...return stack.

15. The method of claim 12 comprising a specific task control block assigned to

each task, wherein a single task-resume address is saved.

16. The method of claim 15 wherein additional task-specific information is saved.

17. The method of claim 15 wherein a task-resume address is obtained in a context switch by placing a label at the point where the task is to resume, and obtaining the address of the label and storing that address as the task-resume address.

18. The method of claim 17 wherein multiple labels are used within a single task to accomplish multiple context switches.

19. The method of claim 11 further comprising a wait-on-event function characterized in that the function is called only at task-level, returns a value based on whether an event is available or not, and initiates a context switch or not based on the returned value.

20. The method of claim 11 further comprising a wait-on-event function enclosed within a (while) loop at task level, and characterized in that the task calls the wait-on-event function in the loop and examines its return code, exiting the loop if the event is available and initiates a context switch if not, and in the event of a context switch, the task recalls the wait-on-event function after resumption, being still in the loop, and repeats this procedure until exiting the loop.